Computing

# Lesson 4: Making Connections

## Part 2

**Physical Computing**

Allen Heard

[1] *Materials from the Teach Computing Curriculum created by the National Centre for Computing Education*

# Task 1 - Melodies

**Copy** the incomplete program below in your development environment.

```
1 from microbit import *
2 import music
3 music.play(music.          )
```

**Select** one of the melodies opposite and **complete** line 3.

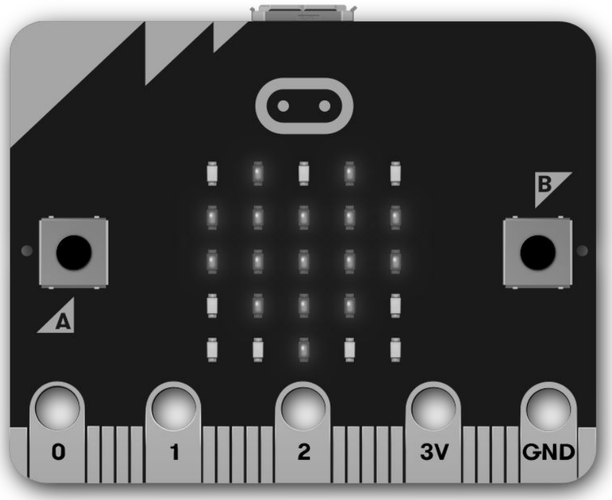
**Flash** the program to your micro:bit, to see it (and hear it) run.

- DADADADUM
- PRELUDE
- NYAN
- FUNK
- BIRTHDAY
- FUNERAL
- PYTHON
- CHASE
- WAWAWAWAA
- POWER_UP
- ENTERTAINER
- ODE
- RINGTONE
- BLUES
- WEDDING
- PUNCHLINE
- BADDY
- BA_DING
- JUMP_UP
- POWER_DOWN

# Task 1 - Output - part 1

In lesson 1, we used the LEDs to display a messages and images.

In this task, you will display images using a loop.




Credit: micro:bit Foundation

# Task 1 - Output - part 1

Replace the three blocks in lines 2, 3, and 4 with images such as HAPPY, SAD, MEH.

You can find the complete list by searching on the internet for *micro:bit images*.

Then add the correct variable name to line 6.

In your development environment, **complete** the program below, so that the list of images contains three built-in `images` of your choice.

```
1  from microbit import *
2  images = [              ,
3                          ,
4                          ]
5  for image in images:
6      display.show(              )
```

**Don't** forget to flash your program.

# Task 1 - Output - part 2

In part 1, the micro:bit displays the images so fast that you only get to see the last image in the list.

Add the line opposite to your program, to introduce a delay between displaying the images in the list.

**Tip:** You may need to indent the new line.

```
+   sleep(1000)
```

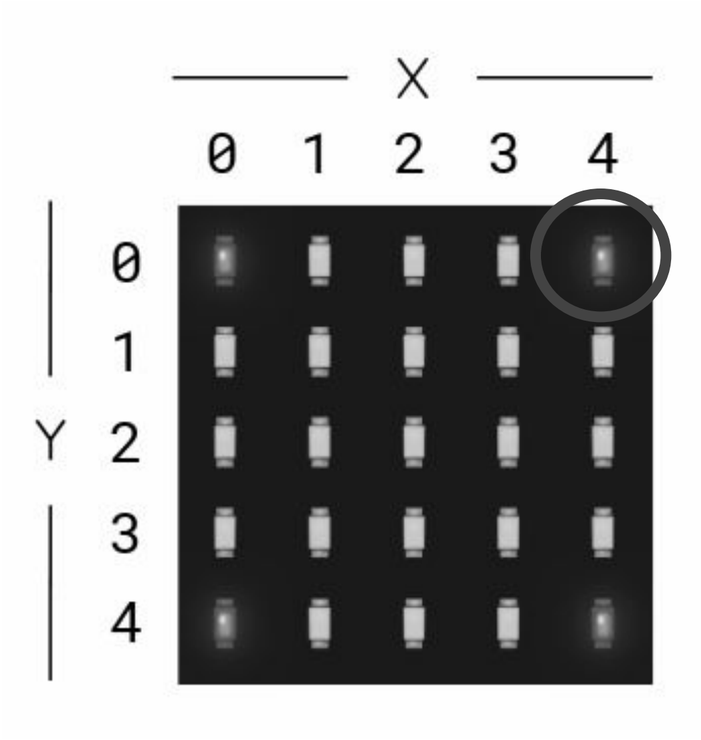
**Note:** The sleep function on the micro:bit takes a number of milliseconds as an argument. Therefore, a 1000 milliseconds is a delay of 1 second. Also, you don't need to import any module to use the sleep function.

# Task 2 - Controlling individual LEDs - part 1

**Complete** the program below to **light up the top right pixel**. You can use the image opposite to work out the x and y coordinates or experiment!

```
1  from microbit import *
2  delay = 100
3  # top right
4  display.set_pixel(■, ■, 9)
```

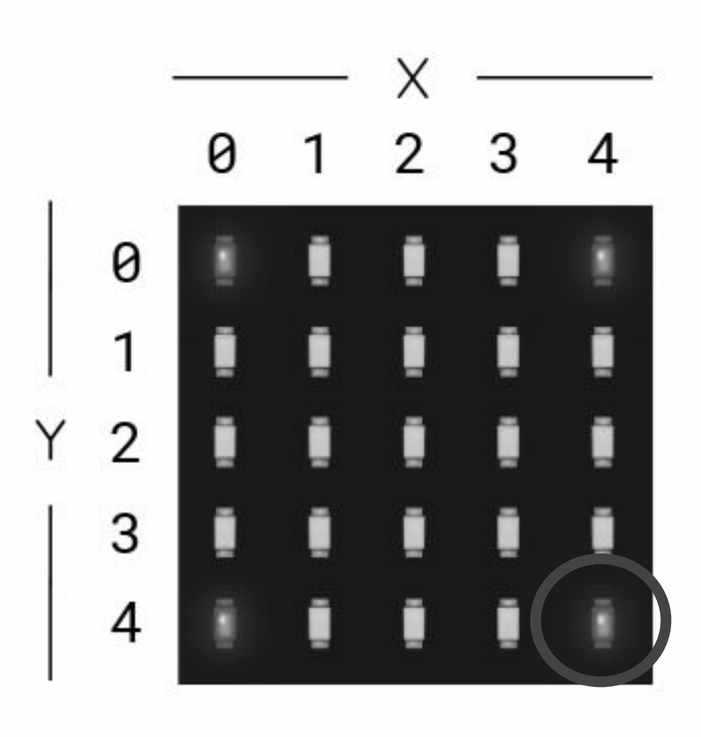
Then, using the worksheet, complete the rest of the parts to this task.



# Task 2 - Controlling individual LEDs - part 2

Add the lines below to your program
**to light up the bottom right pixel**.

```
+   sleep(delay)
+   display.clear()
+   # bottom right
+   display.set_pixel(■ , ■ , 9)
```

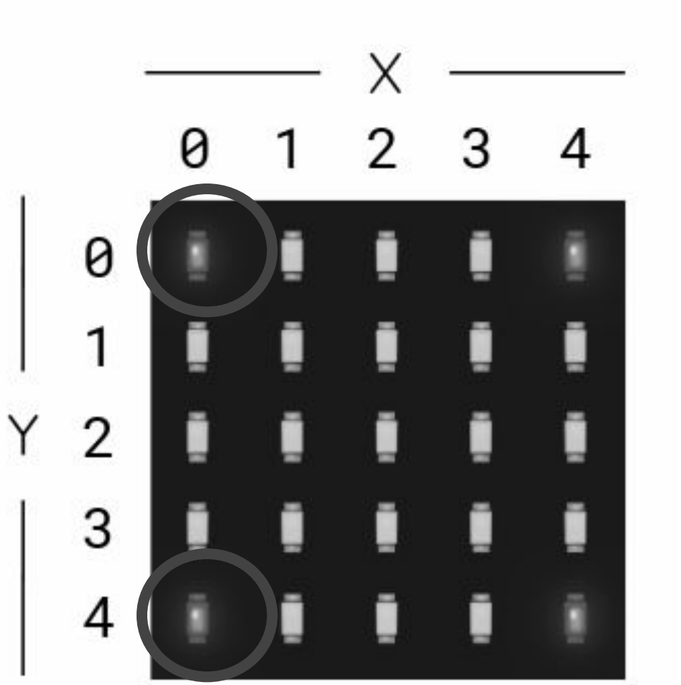


# Task 2 - Controlling individual LEDs - part 2

Repeat part 2, in order to light up the **bottom left** and **top left** pixels (in that order).

Then, nest all of your code in a `while` loop, so that four pixels keep lighting up, one after another forever.

```
while True:
    …
```



# Task 3 - Controlling individual LEDs - part 1

What values would be required to produce the star output shown opposite?

And what other code would you need to add to make it work?

```
star = Image("?????:"
             "?????:"
             "?????:"
             "?????:"
             "?????")
```

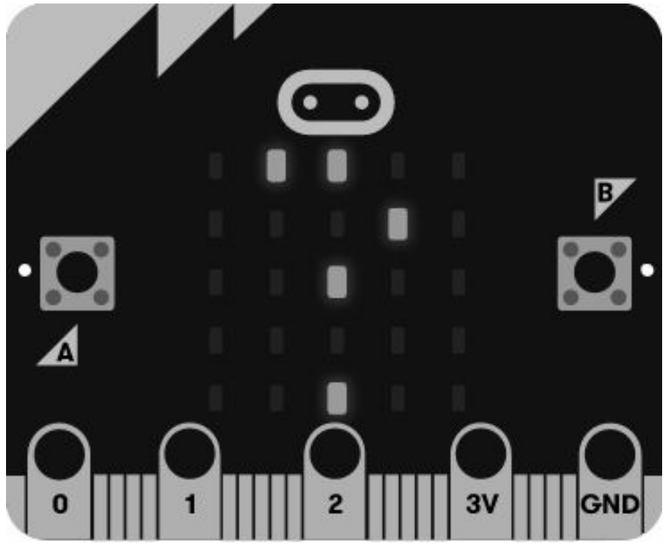
Replicate the image below by filling in the appropriate values for column, row, and brightness.

**Flash** your program to see you results, you may have to adjust your values slightly to make it look the same.

# Task 3 - Controlling individual LEDs - part 2

Create a new program to create your own image.

**Flash** the program to your micro:bit, to see your image displayed.




Credit: micro:bit Foundation

# Task 4 - Sparkle challenge

Complete the program below to set random pixels to a random brightness level and create a sparkling effect on the micro:bit display.

```
from microbit import *
from random import randint
while True:
    x = randint(?,?)
    y = randint(?,?)
    brightness = randint(?,?)
    display.set_pixel(x, y, brightness)
```